Laboratoire d'Électronique numérique:
Utilisation du *PIC16F877*

Année académique 2006-2007

Avant toute chose

1. Créez le répertoire **C :\ELEN040\XX_PIC** où XX est le nom de votre groupe
2. Copiez-y le contenu du repertoire **C :\ELEN040\PIC**

Vous devez faire toujours attention à

1. la mise sous tension de la carte :
   - Alimenter l'ICD2 MAIS PAS la carte.
   - Lancer MPLAB.
   - Choisir *Debugger→Select Tool→MPLAB ICD2.*
   - Dans le menu *Debugger* de MPLAB, choisir l'option *Connect.*
   - Après établissement de la communication, aller dans *Debugger→Settings.*
   - Dans la boîte de dialogue, choisir l'onglet *Power* et vérifier que la case *Power target circuit from MPLAB ICD2* est bien décochée, et appuyer sur OK.
   - Alimenter la carte et choisir *Debugger→Connect.*
2. la sélection du PIC via `#include`"*.inc" Les fichiers *.inc (assembleur) se trouvent dans le répertoire C :\Program Files\Microchip\MPASM Suite\Les fichiers *.h (langage C) se trouvent dans le répertoire C :\C :\HT-PIC\include
3. la configuration du PIC (ligne débutant par __CONFIG), prenez l'habitude de configurer TOUS les bits !
4. la configuration de MPLAB (build options,...)

1 Introduction

Dans ce laboratoire, vous serez amenés à appliquer les concepts développés en répétition et dans l'*Introduction aux microcontrôleurs et à leur assembleur.* Tous les problèmes que vous pourriez rencontrer lors de ces manipulations peuvent être résolus en consultant ces deux références.
RAPPEL important
Tous les composants électroniques doivent être maniés avec délicatesse et précaution. En outre, une attention toute particulière doit être portée à l'électricité statique, qui risquerait d'endommager les circuits. Dans cette optique, nous vous demandons de toujours bien vouloir vous décharger en touchant une prise de terre. Évitez également de toucher les broches des circuits intégrés.

1.1 Sur le PIC utilisé

Le PIC utilisé dans ce laboratoire est un *16F877* (décrit brièvement à la fin de l'*Introduction aux microcontrôleurs et à leur assembleur*). Il s'agit en fait du modèle livré avec la carte de développement. En outre, c'est certainement celui que vous utiliserez dans le cours de Systèmes Programmés Enfouis de M. Boigelot. Il n'est donc pas inutile de déjà s'y familiariser.

Cependant, le niveau des exercices proposés dans ce laboratoire reste assez bas, de sorte que les commandes du *16F84* suffisent amplement pour les réalisé de façon simple.

Il est cependant conseillé de relire la section *7. Evolution : vers le 16F87x...* de l'*Introduction aux microcontrôleurs et à leur assembleur*.

1.2 Sur la carte de développement

Pour ce laboratoire, le PIC est placé sur une carte de développement de type *PICDEM*^{TM} 2 *plus* de chez *Microchip*.

Une **carte de développement** est un ensemble de circuit intégrés précablés entre eux qui permet de réaliser des montages à diverses fins (pédagogiques, test...) sans devoir concevoir un circuit extérieur.

La carte que vous utiliserez dans ce laboratoire (représentée figure 1) est constituée des éléments suivants :

1. Support DIP[1] 18-, 28- et 40-broches. (Bien qu'il y ait 3 supports, un seul composant peut être utilisé à la fois)
2. Régulateur +5V embarqué pour alimentation direct depuis une entrée 9V, 100mA AC/DC (adaptateur secteur), une pile de 9V.
3. Connecteur RS-232 et hardware associé pour connexion directe à une interface RS-232.
4. Connecteur pour ICD[2].
5. Potentiomètre de 5KΩ pour les appareils avec entrée analogique.
6. Trois boutons-poussoir pour générer des stimuli extérieurs et le RESET.
7. LED verte indiquant la mise sous tension.
8. 4 LEDs rouges connectées au PORTB.
9. Jumper J6 pour déconnecter les LEDs du PORTB.
10. Oscillateur à quartz de 4 MHz (prêt à l'emploi).
11. Trous libres pour connecter le cristal.
12. Cristal à 32.768KHz pour les opérations d'horloge du Timer1.
13. Jumper J7 pour déconnecter l'oscillateur RC intégré (fréquence approximative : 2 MHz).
14. 256K x 8 Serial EEPROM.
15. Écran LCD.
16. Buzzer piezo.

[1] *Dual In-line Package*, technique de packaging de composant microélectronique où les broches de connexion, espacées de 2,54 mm, sont alignées sur deux rangées placées le long de chaque côté.

[2] In-Circuit Debugger

17. Aire de prototypage pour le hardware utilisateur.
18. Senseur thermique Microchip TC74.

FIG. 1: *PICDEM 2 Plus* Hardware

1.3 Sur l'ICD2

Comme vous pouvez le voir, la carte de développement possède un connecteur pour raccorder un débogueur en circuit (ICD). Dans ce laboratoire, vous utiliserez MPLAB ICD2, un ICD produit également par Microchip ; et dont l'interface software s'intègre entièrement dans MPLAB IDE.

Comme son nom l'indique, un ICD est un composant électronique (hardware) qui permet (entre autre) de réaliser du débuggage (c'est-à-dire de la correction de code) lorsque le PIC est déjà programmé et inséré dans le circuit. Cette fonctionnalité ressemble un peu à la fonction "Debug" de MPLAB ; mais l'utilisation d'un ICD offre d'autres avantages non négligeables.

En effet, si la simulation du code non implémenté est nécessaire dans une première étape du développement d'un programme ; celle-ci ne peut pas prendre en compte les effets du circuit d'encadrement du PIC dans le monde réél. Or, celui-ci a une grande influence sur les fonctionnalités du PIC. Citons par exemple les problèmes de fonctionnement qui peuvent apparaître lorsque la pin RA4 est laissée flottante en sortie et qu'on lui impose un potentiel extérieur. L'ICD2 permet d'intégrer ces effets dans la simulation puisque le PIC est déjà placé dans son circuit.
L'ICD2 possède également les fonctionnalités d'un programmateur, ce qui évite de devoir renvoyer le PIC sur le DATAMAN (le programmateur universel du laboratoire) à chaque modification.

Au cas où le montage (connexions PC/ICD2/carte de développement/alimentation) ne serait pas raccordé, APPELEZ L'ASSISTANT POUR QU'IL VIENNE LE FAIRE ! ! ! La connexion des composants exige un ordre précis qui, s'il n'est pas respecté, pourrait endommager le matériel.

Un débogage avec l'ICD2 se passe en trois étapes :

1. **Programmation du code pour le débogage**
   L'ICD2 doit, avant de déboguer, placer le code du programme dans la mémoire du PIC. Il charge également un léger code de débogage (environ 0x120 mots) au début de la mémoire programme. Il faut donc que le code du programme n'utilise pas cet espace. Il existe également d'autres restrictions :
   – le mode debug doit être activé ;
   – le watchdog doit être désactivé ;
   – la protection de code dot être désactivée ;
   – la protection de lecture en table doit être désactivée.
   Assurez-vous que TOUS les bits de configuration du PIC soient corrects !
2. **Débogage**
   A cet étape on utilise vraiment le débogueur pour vérifier le bon fonctionnement pas à pas ou par étapes du programme et corriger les éventuelles erreurs résiduelles.
3. **Programmation du code final**
   Une fois le débogage terminé et les erreurs corrigées, l'ICD2 permet de programme le PIC avec le programme final (sans le code de débogage).

Nous explorons plus en détails les capacités de l'ICD2 à la fin du laboratoire.

2 Manipulations

La première partie du laboratoire ne comporte que deux manipulations, c'est-à-dire deux portions de code différentes sur lesquelles vous allez travailler. Le but n'est pas de finir le laboratoire en une demi-heure, mais de repartir en ayant bien compris les concepts de base des microcontrôleurs et de l'assembleur. Prenez donc le temps de bien comprendre à chaque étape ce que vous faites.

Il ne vous est pas demandé ici d'écrire du code. Juste de comprendre le fonctionnement de certaines parties d'un code déjà écrit, ou d'en modifier quelques lignes. Les questions pertinentes à se poser sont donc :

- Que fait cette partie de code ?
- A quoi ces instructions correspondent-elles au niveau hardware ?
- Serais-je capable de modifier ce code pour qu'il réalise l'opération que je désire ?

Les questions et leurs sous-questions sont de difficulté croissante. Si vous n'avez pas compris un exercice, mieux vaut demander de l'aide à l'assistant que de passer à la question suivante : les problèmes risquent de s'accumuler plutôt que de se résoudre !

Sur ces quelques conseils, bon laboratoire, et bon amusement !

2.1 Manipulation 1 :

Cette première manipulation doit vous permettre de vous familiariser avec l'environnement de MPLAB, l'architecture de base du PIC et l'assembleur.

Commencez par vérifier que le jumper J6 est branché et le jumper J7 débranché.
Ensuite Lancez MPLAB IDE. Commencez par créer le projet *Manip1.mcp*. Pour cela, allez dans *Project→Project Wizard*, et choisissez :

- Device : PIC16F877
- Toolsuite : Microchip MPASM
- Project Name : Manip1
- Project Directory : C :\ELEN040\XX_PIC\Led_Cli

Dans la fenêtre de l'espace de travail (*Manip1.mcw*), ajouter *LED_CLI.asm* aux fichiers source (*Source Files*) à l'aide d'un clic droit sur *Source Files* puis en choisissant *Add Files...*. Ensuite, un double clic sur le nom du fichier permet d'ouvrir l'éditeur où vous pourrez modifier le code du programme.

Avant toute chose, commencez par remplir l'entête avec vos informations personnelles. Documenter un programme est un élément très important, il n'est jamais trop tôt pour prendre de bonnes habitudes.

Comme indiqué, ce programme a pour but de faire clignoter une LED sur le port RB1 à une fréquence de 1Hz. La première chose à faire est de **lire le code** dans son entièreté et d'essayer d'en **comprendre** un maximum.
ATTENTION ! Vérifiez que les premières lignes de code (configuration du PIC) correspondent au PIC placé sur la carte. De plus, assurez-vous que l'écriture des variables en mémoires RAM (début de l'allocation mémoire via CBLOCK) est valide. Sinon, modifiez le code.
Lectures conseillées dans l'*Introduction aux microcontrôleurs et à leur assembleur* :

- *2.1.2 Mémoire RAM*
- *2.5 Entrées/sorties*
- *3. Programmation du PIC*

2.1.1 Compilation

Maintenant que vous commencez à comprendre le code, nous allons passer à sa compilation.

- Appuyez sur F10 pour lancer la compilation. Que se passe-t-il ?
- En lisant le message d'erreur renvoyé par le compilateur, corrigez le problème, en ignorant les *Warnings*.
- Relancez la compilation. Y a-t-il encore un problème ?
- Corrigez le nouveau problème.
- Compilez votre fichier corrigé.

2.1.2 Correction de l'erreur

Une fois le code compilé, nous allons l'implémenter dans le PIC pour vérifier son bon fonctionnement. Vérifiez d'abord que l'option *Debugger→Select Tool* est mise sur *None*. Ensuite, choisissez *Programmer→Select Programmer→MPLAB ICD2*. Recompilez le programme pour plus de sécurité.

Notons que les options de compilations peuvent être réglées dans *Programmer→Settings*, mais nous ne nous en soucierons pas ici.

- Programmez le code dans le PIC (*Programmer→Program*). Débranchez l'ICD2 et lancez le programme en appuyant sur le bouton *Reset*. Que se passe-t-il ?
- Essayez de corriger ce problème en modifiant/ajoutant/supprimant une seule ligne dans le code.

2.1.3 Modification du programme

Maintenant, nous allons entrer dans les détails du code.

- Analysez la **routine de temporisation**.
  Indice : le quartz est à 4Mhz et le PIC effectue une instruction tous les 4 cycles d'horloge (8 pour les sauts).
- **Modifiez** la routine de temporisation pour **doubler** la durée de celle-ci.

Si vous avez le temps après cette première manipulation et que vous désirez comprendre encore mieux le fonctionnement du PIC, vous pouvez lancer une simulation pas à pas avec le débogueur interne de MPLAB et observez ce qui se passe dans les principaux registres spéciaux (STATUS, PCLATH, W...). Le fonctionnement de ce débogueur est expliqué dans l'*Introduction aux microcontrôleurs et à leur assembleur*.

2.2 Manipulation 2 :

Cette deuxième manipulation vous amène un peu plus loin dans la compréhension du fonctionnement du PIC. Vous aborderez ici deux de ses particularités : les interruptions et la lecture en EEPROM. La deuxième partie de la manipulation sera également l'occasion pour vous d'avoir un premier contact approfondi avec l'ICD2.

Lectures conseillées dans l'*Introduction aux microcontrôleurs et à leur assembleur* :

- *2.1.3 Mémoire EEPROM*
- *2.4 Timer0*
- *5. Interruptions*
- *6.2 Exemple avancé*

2.2.1 Télérupteur

Pour cette manipulation, il faut débrancher le jumper J6 et connecter une LED sur RB1 (**avec une R de 330 Ohms en série ! !**) ou mesurer la tension de RB1 à l'aide d'un multimètre.
Créez maintenant le projet *Manip2a*, avec les mêmes options, et ajoutez lui *telerupteur.asm* comme fichier source. Le but est de créer un télérupteur, c'est-à-dire un programme qui allume une LED sur une pression d'un bouton et qui l'éteint sur une autre pression.

- **Lisez et comprenez** le code (notamment la **routine d'interruption**). Selon vous, quel est le rôle de la routine `tempo` ?
- Programmez le PIC à l'aide de l'ICD2. Débranchez-le et vérifer le bon fonctionnement du code.
- Rebranchez le jumper J6 et remplacez l'interruption sur RB0 par une **interruption sur Timer0**. La LED ne s'allumera donc plus sur commande mais clignotera à une fréquence dépendant de votre programme.

Pour la deuxième partie de la manipulation, il est recommandé de se référer au deuxième exemple de l'*Introduction...*, ainsi qu'à la section sur le Timer0.

Maintenant, nous allons un peu manipuler l'ICD2 pour mettre en évidence ses possibilités. reprenez le code telerupteru.asm avec interruption sur RB0. Dans MPLAB, sélectionnez *Programmer→Select Tool→None*, puis *Debugger→Select Tool→MPLAB ICD2*. Les diverses fenêtres de débogage sont accessibles via le menu *View*.

- Configurez coorectement le PIC en mode debug.
- Compilez le code et programmez le PIC.
- Commencez par lancer le débogage via *Debugger→Run*, ou F9. Pour arrêter l'exécution du programme, utiliser *Debugger→Halt*, ou F5. Vous pouvez également placer des breakpoint où le programme s'arrêtera et vous aurez tout le loisir de visualiser le contenu des registres et autres variables.
- Il est également possible de faire tourner le programme pas à pas (*Debugger→Step into*, ou F7), ou en exécution animée (*Debugger→Animate*).

3 Utilisation du langage C

Le langage C est utilisé assez fréquemment pour programmer le pic. En effet, il existe différents compilateurs qui permettent d'utiliser les fonctions les plus basiques du langage (PicC-lite, version gratuite qui s'intègre à Mplab).

L'utilisation de celui-ci ouvre beaucoup de perspectives : code beaucoup plus simple et portable, plus besoin de tenir compte du changement de banque, facilité d'accès aux différents bits de configuration (voir annexe : pic16f87x.h), simplicité de l'utilisation des interruptions... Les différentes librairies et les fonctions s'y rapportant se trouvent à la fin du manuel d'utilisation de PicC (ou dans le dossier HT-Soft/PICC/Include).

Néanmoins l'interprétation du code C par le compilateur n'est pas toujours optimale. Il sera parfois préférable de travailler en assembleur pour pouvoir contrôler le nombre d'instruction. Cela dépendra du but de l'application et de la précision qu'elle requiert. Cependant, il est possible d'insérer du code "assembleur" dans le code C, en utilisant les balises #asm...#endasm (pour plusieurs lignes de code) ou asm() (pour l'utilisation d'une seule instruction).

Voici un tableau des formats de variables utilisables :

Choisissez le type de variable en fonction de l'utilisation que vous allez en faire. Prenez garde aux opérations de multiplication et de division.

4 L'écran LCD

Le LCD est connecté au pic via le PORTD et le PORTA. On utilise les 4 bits de poids faible du PORTD pour le transfert de données, les bits 1, 2 et 3 du PORTA, respectivement, pour configurer l'entrée "enable", pour lire ou écrire les données du PORTD et pour sélectionner le "registre" (permet de choisir si on travaille en instruction interne ou affichage/lecture sur l'écran - voir datasheet). Consulter les datasheets en annexe (.\labo_pic_datasheet_LCD.pdf) pour vous familiariser avec les instructions.

Principe de Base (voir schéma datasheet) Le LCD comporte 2 RAM, une RAM qui contient les données affichées (DDRAM) et une RAM qui contient les valeurs ASCII disponible à l'affichage (CGRAM). Les registres les plus importants sont décrits ci-dessous :

4.1 Instruction Register (IR) :

Registre qui va réceptionner le code d'instruction envoyé (Clear display,...) mais aussi dans certains cas une adresse pour la DDRAM (data display RAM = DDRAM), ou une adresse pour la CGRAM (Character Generator RAM).

4.2 Data Register (DR) :

Stocke la valeur qui va être écrite ou lue dans la DDRAM/CGRAM.

4.3 Busy flag (BF) :

Quand le flag est à 1, le système est en mode d'opération interne et l'instruction suivante ne sera pas acceptée.

4.4 Address Counter (AC) :

Le compteur d'adresse assigne l'adresse à la DDRAM ou à la CGRAM en fonction de l'instruction utilisée. Quand une adresse est envoyée dans le registre IR, elle est copiée dans le registre AC. Après avoir été copiée dans la RAM choisie, AC est automatiquement incrémenter ou décrementer de 1 selon l'initialisation.

4.5 Display data RAM (DDRAM) :

Stocke des données représentées par un code caractère de 8 bits. Sa capacité est variable selon le modèle et, est souvent supérieur au nombre de caractère affichable sur l'écran (2x16 caractères dans notre cas).

4.6 Character generator RAM (CGRAM) :

Génere des caractères de 5 x 8 points codés sur 8 bits.

Les fonctions et registres décrits dans les datasheets sont utilisés par la plupart des LCD. Vous pourrez donc utiliser ce code pour un écran de marque différente. Néanmoins, il est toujours nécessaire de vérifier les données techniques fournies par le distributeur.

5 Convertisseur A/D

Le PIC 16F877 dispose d'un convertisseur analogique digitale. Il permet de convertir une valeur Analogique 0-5 V en une valeur numérique pouvant aller jusqu'à 10 bits. Plus vous augmenterez la précision de la mesure et plus le temps de mesure sera élevé. Il faudra donc parfois faire un compromis vitesse-précision.

Le principe de conversion est simple, la tension à numériser est appliquée à un condensateur interne et, une fois le condensateur chargé, on applique le principe de l'approximation successive (voir cours de mesure).

Le pic dispose de 8 entrées analogiques (AN0-AN7) mais un seul convertisseur (ou 2 selon la version). On ne peut donc numériser qu'un signal à la fois (ou 2).

La valeur convertie est placée dans les registres ADRESH (MSB) et ADRESL (LSB).

Les 2 **registres de configuration** du convertisseur sont **ADCON0** et **ADCON1**. Ils permettent :

- de choisir les pins d'entrée qui seront utilisée en analogique (PCFG).
- de choisir la vitesse de conversion.(ADCS1 et ADCS0)
- de choisir le canal sur lequel on va lancer la conversion.(CHS0-CHS2)
- de lancer la charge du condensateur (ADON)
- de lancer la conversion (ADGO)
- de vérifier quand la conversion est terminée (ADGO)
- de configurer la manière de représenter les données des valeurs converties dans ADRESH et ADRESL (ADFM)

6 Manipulations

6.1 Manipulation 3

Dans Mplab, ouvrez le projet *interrupt.mcp* se trouvant dans le dossier .\Interrupt. Il s'agit du programme de télérupteur implémenté en langage C.

- Comparez les 2 programmes (telerupteur.asm et telerupteur.c). Analyser les 2 routines d'interruption et trouver les différences les plus marquantes.
- Changez la routine d'interruption pour effectuer une interruption sur le timer0 (Attention on ne peut utiliser une même fonction dans le main et dans la routine d'interruption, il faudra donc implémenter des fonctions spéciales pour l'interruption : par ex DelayMs_interrupt dans delay.c)

6.2 Manipulation 4

Ouvrez le projet *lcd.mcp* dans le dossier .\lcd. Comme son nom l'indique, ce petit programme vous montre comment gérer un écran LCD.

6.2.1 Initialisation

Ouvrez le fichier lcd.c et, analyser les différentes fonctions définies dans celui-ci. L'initialisation est décrite dans les datasheets (les délais sont donnés par le constructeur). Comprenez bien chacune de ces lignes, elles permettent de configurer l'écran.

6.2.2 Manipulation

- Commentez chaque ligne de la fonction lcd_init().
- Familiarisez-vous avec les différentes fonctions, en n'oubliant pas de commencez par lcd_init().
- Maintenant que vous maîtriser bien ces fonctions, effectuez les tâches suivantes (Entre chaque fonction, utilisez la fonction DelayS(2); pour marquer les différentes étapes de votre application) :
  - Initialisez le curseur clignotant (« blink cursor »)
  - Afficher le nom d'un membre de votre groupe sur la 1ere ligne
  - Afficher le nom de votre section sur la 2ème ligne
  - Rajouter le prénom d'un membre de votre groupe à la suite de votre nom de section, en utilisant la fonction lcd_goto().
  - Ajoutez les lignes de code suivantes et compilez :

```
« DECLARER s AU DEBUT DU MAIN : unsigned char s ;
lcd_goto(0x10);
lcd_puts("Phrase Cachée");
lcd_home();
lcd_write(0x1C);

while(s<27)
{
lcd_write(0x18);
  DelayS(1);
  s++;
```

```
}
```

– Expliquez ces lignes de codes, et adaptez ensuite celle-ci pour déterminer la taille « réelle » de l'écran c'est-à-dire la DDRAM.

6.3 Manipulation 5

Ouvrez le projet *a2d.mcp* dans le dossier .\a2d.

Nous n'allons pas entrer dans des détails trop théoriques mais l'analyse du fichier a2d.c permet de comprendre assez facilement les principes d'initialisation et de mesure (pour plus d'infos, consultez les datasheets et/ou les notes de Bigonoff disponibles sur internet).

Le programme main.c utilise l'entrée analogique AN0 qui est reliée à un potentiomètre (RA0) permettant de faire varier une résistance de 0 à 5k ohms et, donc la tension en entrée de 0 à 5V.

- Analyser le fichier a2d.c
- Lancez la compilation et, exécutez l'application
- Analyser et commentez les lignes de code suivantes :
  ```
  converted=ADRESH;
  converted=(converted << 8);
  converted|=ADRESL;
  ```
- Analyser et commentez les lignes de code suivantes :
  ```
  volts=((n-1)*volts+converted)/n;
  ```
- Quel est le but de l'application ? Expliquez l'utilité du bouton RB0.
- Quelle est la valeur maximum appliquée en entrée ?
- Quelle est la valeur minimum ?
- Cette valeur représente-t-elle des volts comme son nom pourrait le faire croire ?
- A partir des valeurs max et min, déterminez le nombre de bit sur lequel la mesure est convertie.
- Imaginez une fonction permettant d'afficher la valeur acquise en volts.